# Petri nets with semi-structured data

Eric Badouel[1], Loïc Hélouët[1], Christophe Morvan[1,2]

1: INRIA Rennes Bretagne Atlantique
2: Université Paris-Est
eric.badouel,loic.helouet@inria.fr, christophe.morvan@u-pem.fr

**Abstract.** This paper proposes Structured Data Nets (SDN), a Petri net extension that describes transactional systems with data. In these nets, tokens are semi-structured documents. Each transition is attached to a query, guarded by patterns, (logical assertions on the contents of its preset) and transforms tokens. We define SDNs and their semantics. We then consider their formal properties: coverability of a marking, termination and soundness of transactions. Unrestricted SDNs are Turing complete, so these properties are undecidable. We thus use an order on documents, and show that under reasonable restrictions on documents and on the expressiveness of patterns and queries, SDNs are well-structured transition systems, for which coverability, termination and soundness are decidable.

## 1 Introduction

Web services and business processes are now widely used applications. Many solutions exist to design such systems, but their formal verification remains difficult due to the tight connection of workflows with data [19, 14, 25]. For instance, in an online shop one faces situations where a workflow depends on data (*if the age of the client is greater than* 50*, then propose service* $S$), and conversely data depend on a flow (*return an offer with the minimal price proposed among the 5 first values returned by sub-contractors*). These systems have to be open: they must accept user inputs and manage multiple concurrent interactions. Openness also raises robustness issues: a system must avoid interferences among distinct transactions, and be robust for all inputs, including erroneous or obfuscated ones. Last, a transactional system usually manages its own data: catalog, clients database, stock,... which contents influences the execution of the current transactions.

Thus, exact descriptions of transaction systems lead naturally to infinite state models with infinite data and zero tests, that can be captured only by Turing powerful formalisms for which verification problems are undecidable. As a consequence, one has to work with abstractions of these systems to apply automated analysis techniques. Coarse grain approximations can rely on finite discretizations of data or on bounds on the number of transactions in a system. These straightforward techniques allow one to get back to the familiar models of finite state systems or (variants of) Petri nets for which verification techniques

are well-studied and decidable (model-checking for automata, coverability and reachability techniques for Petri nets). However, such bounded discretization that completely abstracts from data is usually too coarse.

This paper introduces *Structured Data nets* (SDN), a variant of Petri nets where tokens are (semi-structured) documents, and transitions transform data. A token represents a piece of information that either belongs to a database associated with the system, or is attached to some ongoing transaction. Each transition of an SDN is attached a query, that is used to transform data, and is guarded by patterns expressing constraints on tokens in its input places. When firing a transition, the corresponding input documents are consumed and new documents computed as the result of queries applied to the input documents are produced in its output places. Fresh data are introduced in the system using an input transition that non-deterministically produces new documents corresponding to new transactions. Termination of a transaction is symbolized by the consumption of a document by an output transition. We define semi-structured documents as trees whose nodes carry information given by lists of attributes/values (*à la* XML). We show that considering documents of bounded depth labeled by well-quasi ordered values, one can provide a well-quasi ordering on documents. We define SDNs and their semantics, and we consider formal properties of this model, such as coverability of a marking, termination and soundness of transactions. In their full generality, SDNs are Turing complete, so all these properties are undecidable. However, we prove that as soon as SDN manipulate well-quasi ordered documents, and meet some reasonable restrictions on the expressive power of patterns and queries (monotonous with respect to ordering), SDNs are well-structured transition systems. If in addition an SDN meets effectiveness requirements, well-structure yields that coverability of a marking is decidable. As a consequence, termination and soundness are also decidable. All these properties hold for a single initial marking of a net, but can be extended to handle symbolically unbounded sets of initial markings satisfying constraints defined by a pattern. Even if some information systems can not be represented by these well-structured SDNs, this decidable setting lays at a reasonable level of abstraction: it does not fix an *a priori* bound on the number of transactions, nor impose finiteness of data values.

Our model borrows elements from Petri nets, but also from data-centric models such as AXML [2] and business artifacts [22]. It is not the first extension of Petri nets which handles complex types attached to tokens: Petri nets with token carrying data have been proposed by [19]. For this extension, coverability of a configuration is decidable. However, data is not really transformed through the workflow, and is mainly used to adapt the structure of flows of an affine nets at runtime. Colored Petri nets [18] can also be considered as Petri nets with data. However, it is well-known that colors give a huge expressive power to nets, and can be used to encode arithmetic operations. It is hence hard to find a reasonable syntactic subclass of colored nets that is amenable to verification. Yet, our model could be simulated with complex coloring mechanisms. Several formalisms handling data have also been designed outside the Petri net com-

munity. **Programming languages** such as BPEL [6] and ORC [20] have been proposed. BPEL is the de facto standard to design Business processes. A BPEL specification describes a set of independent communicating agents. Coordination is achieved through message-passing. Interactions are grouped into sessions implicitly through *correlations*, which specify data values that uniquely identify a session—for instance, a purchase order number. ORC [20] is a programming language for the orchestration of services. It allows algorithmic manipulation of data, with an orchestration overlay to start services and synchronize their results. **Data-centric approaches** such as *Active XML* (AXML) [2] or *tree pattern rewriting systems* (TPRS) [14] define web services as a set of guarded rules that transform semi-structured documents described, for instance, in XML. They do not make workflows explicit, and do not have a native notion of transaction either. To implement a workflow in an AXML specification, one has to integrate explicitely control states to AXML documents, guards and rules. Decidability of coverability has been proved for the subclass of "positive" AXML [3], in which rules can only append data to a document, and for TPRS manipulating documents of bounded depth. **Artifact-centric approaches** such as *business artifacts* [22] describe the logic of transactions for systems equipped with databases. The workflow of a transaction is defined using automata, or logical rules. A transaction carries variables, which are instantiated by values collected along the workflow or entered by the user. Verification of business artifacts has been proved feasible in a restricted setting [10]. In their original version, business artifacts only consider sequential processing of cases. They have inspired *Guard Stage Milestones* (GSM) [17], that allows parallelism among tasks. Recently we have introduced a grammar based artifact-centric case management system [7] which enables transparent distribution of tasks.

One can also mention several initiatives to model web-services in the **$\pi$-calculus** community. *Session types* [16] have been proposed as a formal model for web services. The expressive power of the whole $\pi$-calculus and session types do not allow for verification of reachability or coverability properties. [4] uses WSTS to show that a fragment of spatial logic that can express safety properties is decidable for well-typed $\pi$-calculus processes. An effective forward coverability algorithm for $\pi$-calculus with bounded depth has been proposed in [25]. Last, several formalisms such as $\mu$-se [9], CASPIS [8], COWS [24], have been proposed to model web-services.

This paper is organized as follows: Section 2 introduces the basic elements of our model, namely documents and tree patterns. Section 3 shows how documents can be ordered. Section 4 defines Structured Data Nets, and their semantics. We then consider formal properties of this model, and in particular coverability of a marking, termination, and soundness of transactions in Section 5. Section 6 concludes this work and gives future lines of research.

## 2 Documents and Tree patterns

Our model of net is a variant of Petri nets manipulating structured data. These data are encoded as trees, and queried using tree patterns and queries.

A *tree* $T = (V, E)$ consists of a set $V$ of vertices with a distinguished vertex, $\text{root}(T) \in V$, called the *root* of the tree, together with a set of *edges* $E \subseteq V \times V$, such that for every vertex but the root $v \in V \setminus \{\text{root}(T)\}$ there exists a unique path from the root to $v$, i.e. a finite sequence $v_0, \ldots, v_n$ such that $v_0 = \text{root}(T)$, $(v_{i-1}, v_i) \in E$ for $1 \leq i \leq n$ and $v_n = v$. In particular, *(i)* every vertex but the root $v \in V \setminus \{\text{root}(T)\}$ has a unique predecessor, i.e. a vertex $v'$ such that $(v', v) \in E$, and the root has no predecessor. A tree is *labelled* in $A$ if it comes equipped with a labelling function $\lambda : V \to A$. The *depth* of a tree $T$ is the maximal length of a sequence of consecutive edges in $T$.

Tokens of Structured Data Nets are *documents* represented by finite trees whose nodes are labelled with attribute/value pairs, i.e. by a finite set of equations of the form $a = v$ where tag $a$ denotes a data field or an attribute and $v$ its associated value. For that purpose we let a *tag system* $\tau = (\Sigma, \mathbb{D})$ consist of a set $\Sigma$ of tags and a set $\mathbb{D}$ indexed by $\Sigma$ such that for every $\sigma \in \Sigma$, the set $\mathbb{D}_\sigma$ of possible values for attribute $\sigma$ is non-empty. A *valuation* $\nu \in \text{Val}_\tau$ associated with a tag system $\tau = (\Sigma, \mathbb{D})$ is a partial function $\nu : \Sigma \to \mathbb{D}$ whose domain of definition, denoted $\text{tag}(\nu)$, is finite and such that $\forall \sigma \in \text{tag}(\nu), \nu(\sigma) \in \mathbb{D}_\sigma$.

**Definition 1 (Documents).** *A* document $D \in \text{Doc}_\tau$ *associated with a tag system* $\tau$ *is a finite tree labelled by valuations in* $Val_\tau$*.*

If $v$ is the node of a document, we let $\text{tag}(v)$ be a shorthand for $\text{tag}(\lambda(v))$ and let $v \cdot \sigma$ denote $\lambda(v)(\sigma)$ when $\sigma \in \text{tag}(v)$. We use *tree patterns* to address boolean properties of trees. A tree pattern is also a labelled finite tree, whose edges are partitioned into ordinary edges and ancestor edges, and whose nodes are labelled by constraints. A *constraint*, denoted by $C \in \text{Cons}_\tau$, is defined by a partial function[1] $C : \Sigma \to \wp(\mathbb{D})$ whose domain, denoted $\text{tag}(C)$, is finite and such that $\forall \sigma \in \text{tag}(C), C(\sigma) \subseteq \mathbb{D}_\sigma$. For instance if $\mathbb{D}_\sigma$ is the set of integers then $5 \leq \sigma \leq 20$ constrains the value of $\sigma$ to lay within the set of integers ranging between 5 and 20, and $\sigma =?$ allows $\sigma$ to take value in the whole set of integers.

**Definition 2 (Tree Pattern).** *A tree pattern,* $P \in \text{Pat}_\tau$*, is a tuple* $P = (V, Pred, Anc, \lambda)$*, where* $Pred, Anc \subseteq V \times V$ *are disjoint set of edges and* $(V, Pred \cup Anc, \lambda)$ *is a finite tree labelled by constraints in* $Cons_\tau$*.*

As for documents, we let $\text{tag}(v)$, for $v$ a node of a tree pattern, be an abreviation for $\text{tag}(\lambda(v))$ and let $v \cdot \sigma$ denote $\lambda(v)(\sigma)$ when $\sigma \in \text{tag}(v)$. We further let $v \cdot \sigma =?$ as an shorthand for $v \cdot \sigma = \mathbb{D}_\sigma$ which means that $v$ must carry the tag $\sigma$ but the value of this tag is not constrained. This situation should not be confused with $\sigma \notin \text{tag}(v)$ which does not constrain node $v$ to carry tag $\sigma$ (see Figure 1 for an illustration). Pattern satisfaction is formally defined as follows:

[1] $\wp(\mathbb{D})$ denotes the set of subsets of $\mathbb{D}$

**Definition 3 (Pattern Satisfaction).** *A document $D = (V_D, E_D, \lambda)$ satisfies a tree pattern $P = (V_P, Pred, Anc, \lambda_P)$, denoted $D \models P$, when there exists an injective map $h : V_P \to V_D$ such that:*

1. $h(root(P)) = root(D)$,
2. $\forall v \in V_P \quad tag(v) \subseteq tag(h(v))$,
3. $\forall v \in V_P \; \forall \sigma \in tag(v) \quad h(v) \cdot \sigma \in v \cdot \sigma$,
4. $\forall (v, v') \in Pred_P \quad (h(v), h(v')) \in E_D$, *and*
5. $\forall (v, v') \in Anc_P \quad (h(v), h(v')) \in E_D^*$ [2].

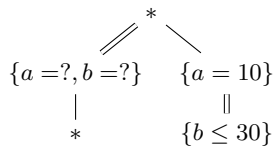



**Fig. 1.** A tree pattern where $* = \{\}$ denotes the empty constraint. It describes the set of trees which have five nodes $v_0, v_1, v_2, v_3$, and $v_4$ with the following properties. $v_0$ is the root of the tree $v_1$ is not a leaf node (i.e. it has at least one successor node $v_2$) and it carries tags $a$ and $b$ ($\text{tag}(v_1) \supseteq \{a, b\}$) with no particular constraints on their values: $\lambda(v_1)(a) = \mathbb{D}_a, \lambda(v_1)(b) = \mathbb{D}_b$. Node $v_3$ is an immediate successor to the root, it carries tag $a$ ($\text{tag}(v_2) \supseteq \{a\}$) and the value attached to tag $a$ is 10. Node $v_4$ is some successor node of $v_3$ tagged by $b$ and the value attached to $b$ is lower than 30.

## 3 Ordering Trees

We shall not distinguish between two isomorphic trees, i.e. when there exists a bijection $\varphi : V^t \to V^{t'}$ between their respective sets of vertices such that $(v, v') \in E^t \iff (\varphi(v), \varphi(v')) \in E^{t'}$ (and thus also $\varphi(v_0^t) = v_0^{t'}$), and $\lambda(v) = \lambda(\varphi(v))$.

If $(A, \leq)$ is an ordered set (respectively a quasi-ordered set, i.e. $\leq$ is a reflexive and transitive relation) then the set of trees labelled in $A$ can be ordered (resp. quasi-ordered) by setting, for any pair of trees $T_1 = (V_1, E_1, \lambda_1)$ and $T_2 = (V_2, E_2, \lambda_2)$, $T_1 \leq T_2$ when there exists an injective map $f : V_1 \to V_2$ such that:

1. $f(\text{root}(T_1)) = \text{root}(T_2)$,
2. $(v, v') \in E_1 \implies (f(v), f(v')) \in E_2$, and
3. $\forall v \in V_1, \lambda_1(v) \leq \lambda(f(v))$.

Hence $T_1 \leq T_2$ if $T_2$ can be obtained from $T_1$ by adding new edges and/or replacing existing labels by greater ones. For instance, given an order relation, $\leq_\sigma$, on $\mathbb{D}_\sigma$ and a subset of tags, $\Sigma' \subseteq \Sigma$, one obtains a quasi-order on $\text{Doc}_\tau$ associated with the quasi-order on valuations $\text{Val}_\tau$ given by:

$$\nu \leq_{\Sigma'} \nu' \iff \text{tag}(\nu) \cap \Sigma' \subseteq \text{tag}(\nu') \quad \wedge \quad \forall \sigma \in \text{tag}(\nu) \cap \Sigma' \quad \nu(\sigma) \leq_\sigma \nu'(\sigma)$$

Thus, restricted to tags in $\Sigma'$, valuation $\nu'$ has a larger domain and associates greater values to tags for which both $\nu$ and $\nu'$ are defined (see Figure 2 for an illustration). Note that $\Sigma' \subseteq \Sigma'' \implies \leq_{\Sigma''} \subseteq \leq_{\Sigma'}$.

[2] $R^*$ denotes the reflexive and transitive closure of relation $R$.

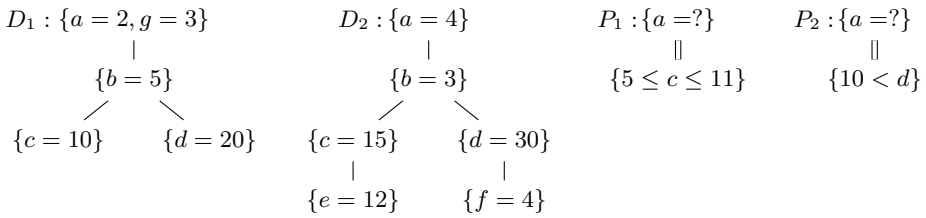



**Fig. 2. Documents and patterns:** Assume all the domains $\mathbb{D}_\sigma$ are given by the set $\mathbb{N}$ of natural numbers with their usual ordering, then $D_1 \leq_{\{a,c,d\}} D_2$. Pattern $P_1$ is not monotonous since e.g. $D_1 \leq_{\{a,c\}} D_2$, $D_1 \models P_1$ and $D_2 \not\models P_1$. By contrast pattern $P_2$ is monotonous.

**Definition 4 (Monotony).** *A pattern $P$ is* monotonous *if $D_1 \leq_{\Sigma'} D_2$ and $D_1 \models P$ implies $D_2 \models P$ where $\Sigma'$ is the set of tags occurring in $P$.*

As illustrated in Figure 2, a pattern that imposes upper bounds on attribute values is not monotonous.

Let us recall classical notions related to orders. A *well quasi-order* (wqo) is a quasi order that is *well-founded*: any infinite sequence $x_1, \ldots, x_n, \ldots$ contains two elements $x_i$ and $x_j$ such that $i < j$ and $x_i < x_j$. Equivalently, a quasi order is a wqo if it contains no infinite strictly decreasing sequences nor infinite antichains (sets of pairwise incomparable elements). Let $\uparrow x = \{y \mid x \leq y\}$ denote the *upward closure* of an element $x$. A set $X$ is upward closed if $\uparrow X = X$, and any upward closed set in a wqo has a *finite* basis, namely a set $B(X) \subseteq X$ such that $\bigcup_{x \in B(X)} \uparrow x = X$. This property ensures the existence of a finite representation for infinite upward closed sets of elements. Finding a wqo on structured data can serve to finitely represent collections of data of arbitrary sizes, or to allow symbolic manipulations on families of trees. However, in contrast with Kruskal's theorem, which states that tree *embedding* is a well quasi-order on the set of finite trees, the set $(\mathrm{Doc}_\tau, \leq_{\Sigma'})$ is in general not a wqo even if the set of tags is finite and their domains are finite or well quasi-ordered. The reason is that the order relation used here is a *strict rooted inclusion*, which allows construction of sets of pairwise incomparable elements of arbitrary sizes (as shown in Figure 3).

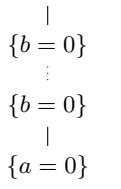



**Fig. 3.** Let us consider tag system $\tau = (\{a, b\}, \mathbb{D})$, with $\mathbb{D}_a = \mathbb{D}_b = \{0\}$ and the tree shown next, denoted $a.b^k.a$, whose root $v_0$, tagged $a$ with $\lambda(v_0)(a) = 0$, is followed by a sequence $v_1, \ldots, v_k$ of nodes tagged $b$ with value $\lambda(v_i)(b) = 0$, and ends with a node $v_{k+1}$ tagged $a$, with $\lambda(v_{k+1})(a) = 0$. The set of trees $\{a.b^k.a \mid k \in \mathbb{N}\}$ consists of pairwise incomparable elements for $\leq_{\{a,b\}}$, hence they form an infinite antichain, whereas they form a chain for tree embedding.

This problem (existence of sets of pairwise incomparable elements of arbitrary sizes) can be avoided by restricting to trees of bounded depth. Let us denote $\mathrm{Doc}_{\tau,\leq n}$ the set of documents whose depth is less or equal to $n$. In order for $(\mathrm{Doc}_{\tau,\leq n}, \leq_\Sigma)$ to be a wqo one must also assume that the set of tags, $\Sigma$, is

finite. If it is not the case, the family of trees reduced to their root and all labelled with distinct tag whould constitute an infinite antichain.

**Proposition 1.** *Let $\tau = (\Sigma, \mathbb{D})$ a tag system where $\Sigma$ is a finite set, $\Sigma' \subseteq \Sigma$, and $n \in \mathbb{N}$. If, for all $\sigma \in \Sigma'$, $(\mathbb{D}_\sigma, \leq_\sigma)$ is a wqo then $(\mathrm{Doc}_{\tau,\leq n}, \leq_{\Sigma'})$ is a wqo.*

*Proof.* First, note that since two documents that only differs on tags that do not belong to $\Sigma'$ are equivalent for the equivalence relation induced by the quasi-order $\leq_{\Sigma'}$, one can assume without loss of generality that $\Sigma' = \Sigma$. We know by [11] that the set of graphs $\mathcal{G}^n_\Sigma$, of bounded depth labelled by well-quasi ordered tags, and ordered by strict subgraph inclusion $\leq$ is a well-quasi order. Therefore the same result holds for trees of bounded depth labelled by wqo, ordered by rooted strict subgraph inclusion $\leq^r$. Indeed one has $T \leq^r T' \iff \overline{T} \leq \overline{T'}$ where $\overline{T}$ is obtained from $T$ by adding a node labelled with a new symbol and by adding an edge from this node to the root of $T$. This additional node is the root of $\overline{T}$ and any strict labelled-graph embedding from $\overline{T}$ to $\overline{T'}$ necessary relates their roots (because of their common label which does not appear elsewhere) and therefore also their unique successor nodes, i.e. the roots of $T$ and $T'$. So it remains to prove that the order relation $\nu \leq_\Sigma \nu' \iff \mathrm{tag}(\nu) \subseteq \mathrm{tag}(\nu') \quad \wedge \quad \forall \sigma \in \mathrm{tag}(\nu) \quad \nu(\sigma) \leq_\sigma \nu'(\sigma)$ on valuations $\mathrm{Val}_\tau$ is a wqo. This order relation can be expressed as: $\nu \leq_\Sigma \nu' \iff \forall \sigma \in \Sigma' \quad \nu(\sigma) \leq^\perp_\sigma \nu'(\sigma)$ where a valuation is viewed as a function $\nu : \Sigma \to \mathbb{D} \cup \{\perp\}$ where $\perp$ is a new element added to each of the sets $\mathbb{D}_\sigma$ as a least element ($x \leq^\perp_\sigma y \iff x = \perp \vee x \leq_\sigma y$) and by letting $\nu(\sigma) = \perp \iff \sigma \notin \mathrm{tag}(\nu)$. Then $(\mathbb{D}_\sigma \cup \{\perp\}, \leq^\perp_\sigma)$ is a wqo for every $\sigma \in \Sigma'$. As, the cartesian product of a *finite* family of wqos is a wqo, we have that $(\mathrm{Val}_\tau, \leq_\Sigma)$ is a wqo. □

## 4 Structured Data Nets

Transactions are usually handled as follows: a new case (a structured document) is created, then it follows a workflow, collecting data in the system. When the transaction is completed, the computed values are returned to the caller. During the workflow, several parallel threads may have been created, and a part of the data of the case (client's name, ...) can be stored in the system for later use.

For convenience, we distinguish two particular transitions that are used to initiate and terminate cases. A transition $t_{in}$, with no incoming place, which delivers to the input place $p_{in}$ a token representing a new transaction. A transition $t_{out}$, with no outgoing place, which unconditionally consumes any token from the output place $p_{out}$.

In addition to case management, a model for transactional systems that allow concurrent threads and concurrent management of many cases has to provide encapsulation of transactions. In a web store, for instance, paying a command should not trigger delivery of someone else's items in another transaction. Encapsulation is often implemented by attaching a session number to a case. Formalisms such as BPEL [23] allow for more elaborated mechanisms called *correlations* to filter and group messages sharing commonalities. In general, it is not

useful to remember exactly the identity of a session, nor to order session identities. A mechanism allowing to differentiate distinct sessions suffices. In [5], we have proposed session systems whose configurations are represented as graphs, and sessions as components of these graphs. In *structured data nets* transactions are handled by assigning an identifier to each individual token, thus inducing a partition on the set of tokens. More precisely, a token is a pair $T = (D, id)$ where $D$ is a document, the value of the token, and $id \in \mathbb{N}$ indicates when $id = 0$ that the data $D$ is part of the local database of the system. Otherwise, $id \neq 0$ provides the identifier of the transaction that $D$ belongs to. Thus identifiers of transactions are positive integers.

Roughly speaking, each input arc $(p, t)$ for $p \in {}^\bullet t$ in a structured data net is attached a guard given by a tree pattern $\langle p, t\rangle$. Transition $t$ is enabled in a marking $M$ if in every of its input place $p \in {}^\bullet t$ one can find a token $T_p = (D_p, id_P) \in M(p)$ such that $D_p \models \langle p, t\rangle$ and all non-null identifiers $id_P$ coincide. The latter condition ensures that all the pieces of information, but those belonging to the local database, are concerned with the same transaction. These tokens are then removed from the current marking and some new tokens should be added to each of its output places $p \in t^\bullet$. For that purpose, each output arc $(t, p)$ for $p \in t^\bullet$ is attached a query $\langle t, p\rangle$ that describes how to compute the value of the token(s) to add in place $p \in t^\bullet$ from the vector of input documents $(D_p)_{p\in{}^\bullet t}$ which enabled the firing of the transition. Queries can produce multisets of tokens. We denote by $\mathcal{M}(A)$ the multisets with elements in set $A$. Every $X \in \mathcal{M}(A)$ is a map $X : A \to \mathbb{N}$, where $X(a)$ gives the multiplicity of element $a \in A$ in $X$. We further let $\mathcal{M}_f(A)$ define the set of finite multisets,i.e., the subset of $\mathcal{M}(A)$ which contains the multisets $X$ such that $X(a) \neq 0$ for a finite number of elements $a \in A$.

**Definition 5 (Query).** *An $n$-ary* query $Q : (Doc_\tau)^n \to \wp(\mathcal{M}_f(Doc_\tau))$ *is a function that non-determinically produces a finite multiset of documents from a vector of documents given as input.*

A query is *simple* when it non-deterministically returns an ordinary set: $\mathrm{Im}(Q) \subseteq \wp(\mathrm{Doc}_\tau)$. A query is *deterministic* if it returns a singleton: $\mathrm{Im}(Q) \subseteq \mathcal{M}_f(\mathrm{Doc}_\tau)$. As illutrated in Figure 4, non-deterministic queries can be used to specify non-deterministic choices of the environment. Non-simple queries can be used to produce several documents, and design creation of concurrent threads.

We leave voluntarily the queries underspecified, as our aim is to define generic properties of nets depending on properties of their documents, query language, and flow structure, but abstracting away as much as possible the query language. Several mechanisms have been proposed to query structured data. Standard query languages such as XQuery [27] and Xpath [26] use patterns to extract information from trees, and are usually described formally as tree pattern queries. The definition of structured data nets is as follows:

**Definition 6 (Structured Data Net).** *Let $\tau$ be a tag system. A structured data net, or SDN, is a structure $\mathcal{N} = (P, P_{DB}, T, F, \langle\cdot,\cdot\rangle)$ where $P$ is a set of places, $P_{DB} \subseteq P$ is a subset of places corresponding to the* local database *of the*

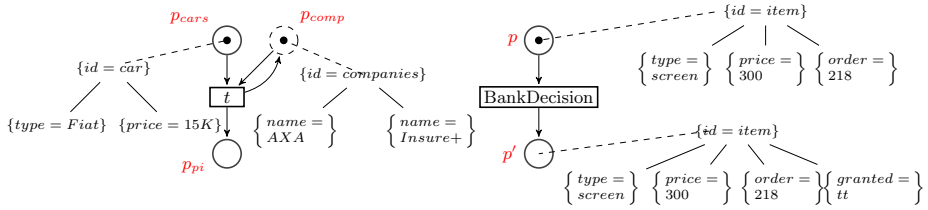



**Fig. 4.** The leftmost example depicts a part of broking system for a car insurance system. Place $p_{cars}$ contains structured documents depicting cars and their price. A token in $p_{comp}$ lists several insurance companies. The place $p_{pi}$ is the starting point to ask pro-forma invoices to companies. The transition $t$ creates one structured document per insurance company that appears in the database, by application of query $\langle t, p_{pi}\rangle$ attached to flow arc from $t$ to place $p_{pi}$. The net on the right models a part of an online shop in which a payment of some bought item needs to be granted by a bank. Transition BankDecision models this decision. The query $\langle \text{BankDecision}, p'\rangle$ attaches a new child to the document's root indicating bank's decision with a boolean. Hence, it non-deterministically returns the input document augmented with either a true or a false boolean tag.

*net, $T$ is a set of transitions, $F \subseteq P\times T \cup T\times P$ is a set of flow arcs. The respective sets of input and output elements of $x \in P \cup T$,* preset *and* postset *are denoted* $^\bullet x = \{y \mid (y,x) \in F\}$ *and* $x^\bullet = \{y \mid (x,y) \in F\}$. *The map* $\langle\cdot,\cdot\rangle : F \to \text{Pat}_\tau \cup \mathcal{Q}_\tau$ *associates each input arc* $(p,t) \in F$ *to a pattern* $\langle p,t\rangle \in \text{Pat}_\tau$ *and each output arc* $(t,p) \in F$ *to an $n$-ary query* $\langle t,p\rangle \in \mathcal{Q}_\tau$ *where* $n = |^\bullet t|$ *is the number of input places of $t$ with a given enumeration of this set of places. SDN possess two places $p_{in}$ and $p_{out}$, and two transitions $t_{in}$ and $t_{out}$ such that* $^\bullet t_{in} = \emptyset$, $t_{in}^\bullet = \{p_{in}\}$, $^\bullet t_{out} = \{p_{out}\}$, $t_{out}^\bullet = \emptyset$, $p_{out}^\bullet = \{t_{out}\}$, *and* $\langle p_{out}, t_{out}\rangle = tt$ *is the trivial pattern reduced to its root labelled with the empty constraint* $* = \{\}$, *i.e. $tt$ is the pattern matched by any document. Any transition such that* $^\bullet t \cap P_{DB} \neq \emptyset$ *has also input places in* $P \setminus P_{DB}$ *ensuring that a transition acts on the database only in the context of the processing of a particular transaction. Finally $t_{in}$ is the unique transition with an empty preset, $t_{out}$ is the unique transition with an empty postset, and any place in* $P \setminus P_{DB}$ *has non-empty preset and postset.*

**Definition 7 (Behaviour of SDNs).** *A* token $T = (D, id) \in \text{Tok}_\tau$ *is made of a document* $D \in \text{Doc}_\tau$ *and a non-negative integer* $id \in \mathbb{N}$. *A* marking $M : P \to \mathcal{M}_f(\text{Tok}_\tau)$ *assigns a finite multiset of tokens to each place such that for all* $(D, id) \in M(p)$ *one has* $id = 0$ *if and only if* $p \in P_{DB}$. *Transition* $t \neq t_{in}$ *is enabled in marking $M$ and firing transition $t$ in marking $M$ leads to marking $M'$, denoted as* $M[t\rangle M'$, *when*

1. $\forall p \in {}^\bullet t \;\; \exists T_p = (D_p, id_p) \in M(p) \quad s.t. \quad D_p \models \langle p,t\rangle$, *and*
   $\exists id \in \mathbb{N} \quad s.t. \quad \forall p \in {}^\bullet t \setminus P_{DB} \quad id_p = id$,
2. $\forall p \in t^\bullet \;\; \exists X_p \in \langle t,p\rangle \left((D_p)_{p\in{}^\bullet t}\right)$,

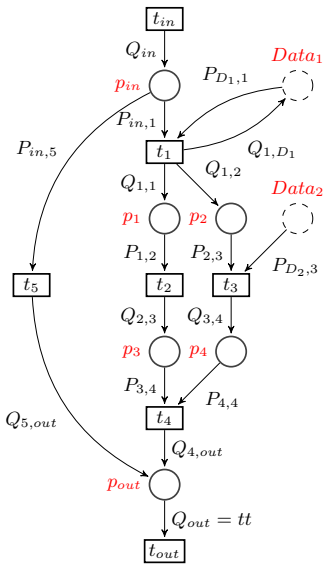



**Fig. 5.** We assume in this example that all queries are simple and all but $Q_{in}$ are deterministic. Then input transition $t_{in}$ creates non deterministically a new transaction by putting a token in place $p_{in}$ containing a document (e.g. a form) together with a new identifier. According to the shape of the token but also to the data contained in place $Data_1$ transitions $t_5$ and $t_1$ may be enabled. For instance $t_1$ may correspond to the nominal behaviour while $t_5$ is used when the document is incomplete or ill-formed. In the latter case the document is immediately transfered to the output place $p_{out}$. In the former case the treatment is split by $t_1$ into two threads (concurrent actions $t_2$ and $t_3$) and the respective results are aggregated by transition $t_4$. Then the output transition $t_{out}$ can withdraw the terminated transaction from the system.

3. $\forall p \in t^\bullet \quad M'(p) = M''(p) \cup \{(D, id_p) \mid D \in X_p\}$ *where* $id_p = id$ *if* $p \notin P_{DB}$ *and* $id_p = 0$ *if* $p \in P_{DB}$, *and* $\forall p \notin t^\bullet \quad M'(p) = M''(p)$; *where* $M''$ *is the marking given by:* $M''(p) = \begin{cases} M(p) \text{ if } p \notin {}^\bullet t \\ M(p) \setminus \{(D_p, id_p)\} \text{ if } p \in {}^\bullet t \end{cases}$.

*The behaviour of transition* $t_{in}$ *is similar except that since it has no input places it is always enabled and no identifier results from the enabling condition. It creates a new identifier associated with the tokens created in input place* $p_{in}$.

When conditions 1 and 2 in Definition 6 are met we say that transition $t$ is *enabled* in marking $M$, denoted $M[t\rangle$. Note that the firing relation $M[t\rangle M'$ is non-deterministic due to the fact that first, one may find several token sets that satisfy the patterns associated with the input places of $t$, and second, the queries associated with the output places may also be non-deterministic. Marking $M'$ is *reachable* from marking $M$ when there exists a sequence of transition firings leading from $M$ to $M'$. We denote $\mathcal{R}(M)$ the set of markings reachable from $M$.

## 5 Properties of Structured Data Nets

The main motivation for using formal notations and semantics is to derive automated tools to reason on the corresponding systems. For transactional systems, one may want to check that a request with correct type is always processed in a finite amount of time, regardless of current data. Another issue can be to guarantee that a payment on an online store is always followed by the sending of the purchased item to the buyer. Last, one may want to check some simple business rules on transactions, confidentiality of some data, etc. In most cases, the properties to check do not deal with global states of the modeled system, but

rather on the status of one particular transaction plus a limited environment. Hence the properties of interest for SDNs are closer to coverability properties than to reachability properties. In this section we formalize and address decidability of reachability, coverability, termination (whether all transactions terminate), and soundness (the question of whether all transactions terminate without leaving pending threads in the system). We can formalize reachability, coverability, termination and soundness as follows for an SDN with respect to a given initial marking $M_0$. We will assume w.l.o.g. that $M_0$ contains no transaction: $\forall p \in P \; \forall (D, id) \in M_0(p) \quad id = 0$.

**Reachability:** Is a given marking $M$ reachable from the initial marking: $M \in \mathcal{R}(M_0)$?

**Coverability:** Is a given marking $M$ smaller than some reachable marking: $\exists M' \in \mathcal{R}(M_0) \;\; s.t. \;\; M \leq M'$?

**Termination of a transaction:** Given a marking $M$ such that a new transaction has just been created ($M(p_{in})$ contains a token $(D, id)$ which is the only token with identifier $id$ in $M$), can one reach a marking $M'$ such that $M(p_{out})$ contains a token $(D', id)$ ? Does one always reach such a marking from $M$ ? Is termination possible or granted for any initial case given by a given marking $M$ or respectively for all initial cases in which the considered document satisfies a given pattern $P$ ?

**Soundness:** Given a marking $M$ such that a new transaction has just been created, does one always reach a marking $M'$ such that $M(p_{out})$ contains a token $(D', id)$ and no other token of the form $(D'', id)$ with the same identifier appears in $M'$?

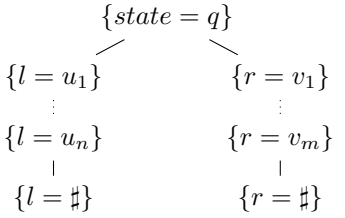
All questions above are undecidable if no restriction is imposed on the nature of documents or queries. In the rest of the section, we consider a class of SDN which is proved to be effective well-structured transition systems, a property that guarantee the decision of coverability.

**Theorem 1 (Undecidability).** *Reachability, coverability, termination and soundness are undecidable problems for SDNs.*

*Proof.* The proof proceeds by a coding of a Turing machine with an SDN. We recall that a Turing machine is made of an infinite bi-directional tape divided in both directions into an infinite number of consecutive cells and a finite state device that can read and write the cell being examined by a read/write head and that can also move that head along the tape in both direction. A cell contains a 0 or a 1, initially every cell has the default value 0. More precisely a Turing machine consists of a finite set of states $Q$ with some initial state $q_0$ and a finite set of instructions of the form $[q, x, \omega, q']$ where $q$ and $q'$ are states, $x \in \{0, 1\}$ is the possible value of the cell, and $\omega \in \{0, 1, L, R\}$ is an operation that corresponds respectively to writing 0 or 1 in the current cell or moving the r/w-head to the left or to the right. A configuration is a triple $(q, u, v) \in Q \times \{0, 1\}^\omega \times \{0, 1\}^\omega$ made of a state $q \in Q$ and two infinite words coding respectively the content of the left part of the tape, read from right-to-left, and the right part of the

tape, read from left-to-right. The r/w-head is positioned on the first cell of the right-part of the tape. The transitions of the Turing machine are given as follows:

1. Writing a value $y \in \{0,1\}$ on the current cell: $(q, u, x \cdot v) \xrightarrow{[q,x,y,q']} (q', u, y \cdot v)$.
2. Right move: $(q, u, x \cdot v) \xrightarrow{[q,x,R,q']} (q', x \cdot u, v)$.
3. Left move: $(q, y \cdot u, x \cdot v) \xrightarrow{[q,x,L,q']} (q', u, y \cdot x \cdot v)$.




A reachable configuration $(q, u, v)$ contains only a finite number of non-null elements therefore one can encode a configuration with a tree as shown next where $\forall i > n, u_i = 0$ and $\forall i > m, v_i = 0$. We let $[q, u, v]$ denote this tree (even though the representation is not unique). In terms of this representation the moves of the Turing machine can be simulated with the rules:

1. Writing a value $y \in \{0,1\}$ on the current cell: $[q, u, x \cdot v] \xrightarrow{[q,x,y,q']} [q', u, y \cdot v]$.
2. $[q, u, x \cdot v] \xrightarrow{[q,x,R,q']} [q', x \cdot u, v]$ and $[q, u, \sharp] \xrightarrow{[q,0,R,q']} [q', 0 \cdot u, \sharp]$.
3. $[q, y \cdot u, x \cdot v] \xrightarrow{[q,x,L,q']} [q', u, y \cdot x \cdot v]$ and $(q, \sharp, x \cdot v) \xrightarrow{[q,x,L,q']} (q', \sharp, 0 \cdot x \cdot v)$.

Each of these rules can straightforwardly be represented by a transition $r$ with ${}^\bullet r = r^\bullet = \{p_{in}\}$ where pattern $\langle p_{in}, r\rangle$ describes those configurations that enable rule $r$ and query $\langle r, p_{in}\rangle$ describes the effect of $r$ on such a configuration. Pattern $\langle t_{in}, p_{in}\rangle = \{[\sharp, q_0, \sharp]\}$ produces the initial configuration. We complete the description of the SDN by adding one transition $halt_{q,x}$ from $p_{in}$ to place $p_{out}$ for each pair of state $q$ and symbol $x$ for which there is no move of the machine of the form $(q, x, -, -)$ where pattern $\langle p_{in}, halt_{q,x}\rangle$ tests that the state is $q$ and the symbol read is $x$ and query $\langle halt_{q,x}, p_{out}\rangle$ witnesses the halting of the Turing machine by creating a specific token, e.g. the empty configuration $[\sharp, q_0, \sharp]$, in the output place. For this SDN reachability or coverability of the final marking with one token in $p_{out}$ are equivalent to termination or soundness thus all these properties are undecidable. □

This result is not surprising, as reachability or coverability are usually undecidable for Petri nets with extended tokens like colored Petri nets. However, one may note several important issues from the encoding of a Turing machine. First, deterministic queries are sufficient for this encoding. Second, three distinct tags and finite domains of values are sufficient to encode a configuration of a Turing machine. An immediate question is whether one can rely on the structure of the data and on simple restrictions to obtain decidability results. A first obvious useful restriction is to bound the depth of documents manipulated by the system. By Proposition 1 the set of documents manipulated by the SDN is a wqo when the domains of the data fields, attached to tree nodes, are wqo. This restriction is reasonable, as it is unlikely that documents grow arbitrarily during their lifetime in a system.

**Definition 8.** *An SDN is well quasi ordered (is a* wqo SDN *for short), when*

*i) the domains of values used by document data fields are well-quasi ordered (finite sets, integers, vectors of integers,...), and comparison is effective (one can can effectively decide if $x \leq_\sigma y$), and*
*ii) there exists a bound on the depth of all trees appearing in $\mathcal{R}(M_0)$.*

Let us comment on the restrictions in Definition 8. Assuming wqo values in documents still allows to work with infinite domains like integers. However, this restriction forbids to attach structured data such as queues of unbounded sizes to nodes. Within the context of transactional systems, this is not a severe limitation. Bounding the depth of documents is not a severe limitation either: Most of transactional systems can be seen as protocols working with a finite number of data fields or using finite forms, in which a finite number of entries needs to be filled. One shall also note that the depth of standard structured documents is usually very low: the structure helps decomposing an entry into data fields, i.e. decomposing a concept into sub-concepts (a person is described as someone with a first name and last name) and it is recognized [21] that 99% of XML documents have depth smaller than 8, and that the average depth of XML documents is 4. Note also that the depth restriction does not mean finiteness of manipulated data: trees of arbitrary width still comply with this restriction, and data values attached to nodes need not be chosen from finite domains. This allows for instance for the manipulation of XML documents containing arbitrary numbers of records. Nevertheless, we show at the end of this section that considering well quasi ordered SDNs is not sufficient to obtain decidability.

Let us define the ordering relation on the set of markings induced by the ordering of documents, and thus ultimately by the ordering of the data values appearing in these documents. The powerset of an ordered or quasi ordered set $(A, \leq)$ is equipped with the quasi order $\leq$ where $X \leq Y$ when an injective map $h : X \to Y$ exists such that $\forall x \in X \quad x \leq h(x)$. For multisets $X, Y \in \mathcal{M}(A)$ we similarly let $X \leq Y \iff |X| \leq |Y|$ where $|X| = \{(x, i) \mid x \in X \wedge 1 \leq i \leq X(x)\}$ denotes the set of occurrences of $X$. Markings are compared componentwise up to an injective renaming of their transactions. More precisely, we let $M_1 \leq M_2$ when there exists an injective map $h : \mathbb{N} \to \mathbb{N}$ such that $h(0) = 0$, and for every place $p$ and every $i \in \mathbb{N}$ one has $\pi_i(M_1(p)) \leq \pi_{h(i)}(M_2(p))$ where $\pi_i(M(p)) = \{D \mid (D, i) \in M(p)\}$ denotes the multiset of documents in $M(p)$ with identifier $i$. As the comparison between two markings is performed up to a renaming of transactions, the exact identifier of a token does not matter. The only concern is whether two tokens with the same (respectively with different) identifier(s) are mapped to tokens with the same (resp. with different) identifier(s). Hence, we can equivalently consider markings as partitions of a multiset [3] of pairs from $P \times \mathrm{Doc}_{\tau,\leq n}$. As a partition of a set $X$ in a set of subsets of $X$, any quasi order on $X$ extends (using twice the powerset extension) to a quasi order on set of partitions of $X$. With this representation $M_1 \leq M_2$ when the two partitions are comparable for the extension to partitions

[3] by partition of a multiset $X$ we mean a partition of the set $|X|$ of occurrences of $X$.

of the ordering $\leq$ on $P \times \mathrm{Doc}_{\tau,\leq n}$ given by $(p, D) \leq (p', D')$ when $p = p'$ and $D \leq D'$.

**Proposition 2.** *The set of markings over bounded depth documents whose data have well quasi ordered domains is a wqo.*

*Proof.* From proposition 1, we know that $(\mathrm{Doc}_{\tau,\leq n}, \leq)$ is a wqo. Since the set of places is finite, the ordering relation on $P \times \mathrm{Doc}_{\tau,\leq n}$ is also a wqo. Last, the product of two wqos forms a wqo [15], and we have seen that extending the ordering to multisets and then to partitions also yields a wqo. Hence, the ordering on markings over documents of bounded depth is a wqo. □

An immediate followup to well quasi orderedness is to set restrictions to obtain well-structured transition systems (WSTS) and reuse existing results to check coverability. An $n$-ary query $Q$ is said to be *monotonous* when

$$(\forall i \in \{1, \ldots, n\} \quad D_i \leq D'_i) \implies Q(D_1, \ldots, D_n) \leq Q(D'_1, \ldots, D'_n).$$

**Proposition 3.** *A wqo SDN with monotonous patterns and queries is a WSTS, more precisely* $(M_1[t\rangle M'_1 \wedge M_1 \leq M_2) \implies (\exists M'_2, M_2[t\rangle M'_2 \wedge M_2 \leq M'_2)$

*Proof.* According to Definition 7 we distinguish the initial transition $t_{in}$, which is responsible for the creation of new identifiers, from the other transitions.
**If $t = t_{in}$:** The transition $t_{in}$ is not guarded, and results in a non-deterministic creation of new documents $D_1, \ldots D_k$ with a fresh identity $id$ in place $p_{in}$, namely $M'_1 = M_1 \uplus \{(p, (D_1, id)) \cup \cdots \cup (p, (D_k, id))\}$. Then, one can find a fresh integer $id'$ that is not used in $M_2$ so that $M_2[t_{in}\rangle M'_2$ where $M'_2 = M_2 \uplus \{(p, (D_1, id')) \cup \cdots \cup (p, (D_k, id'))\}$. As $M_1 \leq M_2$, there exists an injective map $h$ such that for every place $p$ and every $x \in Dom(h)$, $\pi_x(M_1(p)) \leq \pi_{h(x)}(M_2(p))$. We extend this map by letting $h(id) = id'$ to get $\pi_{id}(M'_1(p)) \leq \pi_{id'}(M'_2(p))$ and thus $M'_1 \leq M'_2$.

**General case ($t \in T \setminus \{t_{in}\}$):** This transition is enabled when all the patterns $P_1, \ldots P_k$ attached to flow from places $p_1, \ldots, p_k$ in ${}^\bullet t$ to $t$ are satisfied by some documents $D_1, \ldots D_k$, with the same identifier $id$ for documents located in places ${}^\bullet t \setminus P_{DB}$, and with identifier 0 for documents from ${}^\bullet t \cap P_{DB}$. Upon firing, $t$ consumes $D_1, \ldots D_k$ from ${}^\bullet t$, and outputs a set of newly created documents $D'_1, \ldots D'_{k'}$ with identifier $id$ in places of $t^\bullet \setminus P_{DB}$, and with identifier 0 in places of $t^\bullet \cap P_{DB}$ where $\{D'_1, \ldots D'_{k'}\} = \cup_{p \in t^\bullet} X_p$ for some $X_p \in \langle t, p\rangle(D_1, \ldots, D_k)$. As $M_1 \leq M_2$, there exists an injective mapping $h$ such that for every identifier $x$ and every place $p$, $\pi_x(M_1(p)) \leq \pi_{h(x)}(M_2(p))$. This also yields, for each identifier $x$ and each place $p$ a map $\varphi_{p,x} : \pi_x(M_1(p)) \to \pi_{h(x)}(M_2(p))$, such that $D_i \leq \varphi_{p,x}(D_i)$. Let us denote by $\varphi = \bigcup \varphi_{p,x}$ the union of all these maps for $p \in P$, and $x$ an identifier used in $M_1$

Since guards are monotonous and $D_i \leq \varphi(D_i)$, one has $\varphi(D_i) \models P_i$. From the monotony of queries we deduce that for every place $p \in {}^\bullet t$, there exists $X'_p \in \langle t, p\rangle(\varphi(D_1), \ldots, \varphi(D_k))$ with $X_p \leq X'_p$. Thus transition $t$ is enabled in marking $M_2$ and $M_2[t\rangle M'_2$ with $M'_2(p) = (M_2 \setminus (\{\varphi(D_1), \ldots, \varphi(D_k)\} \cap M_2(p))) \cup X'_p$.

Let us now prove that $M'_1 \leq M'_2$. We can design a set of injective maps $\varphi'_{p,x} : \pi_x(M'_1(p)) \to \pi_{h(x)}(M'_2(p))$ witnessing $M'_1 \leq M'_2$. For every $D_i \in M_1(p) \cap M'_1(p)$,

we define $\varphi'_{p,x}(D_i) = \varphi_{p,x}(D_i)$, as the documents that were not consumed remain unchanged and hence comparable in both markings. Then, for each newly created document $D'_i$ in $X_p$, as $X_p \leq X'_p$, we necessarily have a document $D'_j$ in $X'_p$ such that $D'_i \leq D'_j$. Hence we can set $\varphi'_{p,x}(D'_i) = D'_j$, and obtain $D \leq \varphi'_{p,x}(D)$ for every $D \in M'_1(p)$. Hence, the map $\varphi' = \bigcup \varphi'_{p,x}$ witnesses $M'_1 \leq M'_2$. □

Coverability can be decided using a standard backward algorithm. For a set of markings $X$, we let $pre(X) = \{M \mid \exists t \in T, M' \in X, M[t\rangle M'\}$. We also let $basis(X)$ be a basis for an upward closed set $X$. Let $M$ be the marking that one tries to cover. The algorithm iteratively computes basis for the sets of markings from which a marking in $\uparrow M$ can be reached in a finite number of steps. The algorithm starts from the set $X_0 = \{M\}$, that is a basis for all markings greater than $M$. Then it builds iteratively $X_{i+1} = X_i \cup basis(pre(\uparrow X_i))$, and stops when a fixpoint is reached, or as soon as there exists $M' \in X_i$ such that $M' \leq M_0$, indicating that there exists a sequence of transitions from $M_0$ to a marking greater than $M$. It was proved in [1, 13] that this algorithms is correct and terminates for *effective* WSTS where effectiveness means that *i)* the comparison relation $\leq$ is effective and *ii)* (backward-effectiveness) one can effectively build a finite basis for $pre(\uparrow M)$.

**Corollary 1 (Coverability).** *Coverability is decidable for backward-effective wqo-SDN with monotonous patterns and queries.*

*Proof.* It remains to show that the comparison among markings is effective. For any pair of documents $D_1, D_2 \in \mathrm{Doc}_\tau$, one can effectively check for the existence of a mapping from $D_1$ to $D_2$, and compare the values of paired data fields, as we have assumed that the domains of these datafields are effective wqos. Then finding an identity preserving mapping among contents of places (finite multisets) is also effective. □

Backward effectiveness means that from an upward closed set of markings one can effectively build a finite representation of the data input to a transition that might have generated these contents. This property is easily met if the effect of a transition on a place is to aggregates finite amount of data collected from its input places (for instance the sum of positive integers collected in forms), or to append a new branch to a document (in this case, the input data can be obtained by considering subtrees of the documents appearing in the original marking).

Let us now show that this result on coverability allows to prove more properties. For a pattern $P$, we define $Sym(P)$ the *symbolic set of initial cases* induced by $P$ as the set of documents satisfying $P$. We are now ready to address the termination, soundness, and coverability for symbolic sets of initial cases. The latter coverability question makes sense if one assumes that the query $\langle t_{in}, p_{in}\rangle$ generates $Sym(P)$. This is not always the case, and the set of documents generated by $\langle t_{in}, p_{in}\rangle$ needs not satisfy a single pattern $P$. It may even be the case that this set of initial cases is not upward closed (for instance, a query can generate documents which nodes carry only odd integer values). The coverability problem for the set of initial cases induced by $P$ can be rephrased as follows:

assuming $IMG(\langle t_{in}, p_{in}\rangle) = Sym(P)$, and given a marking $M$ to cover, can one find an initial marking $M_0$ such that $M_0(p_{in}) \in Sym(P)$ and there exists $M'$ greater than $M$ in $\mathcal{R}(M_0)$ ?

**Theorem 2.** *Termination, soundness, and coverability for symbolic set of initial cases defined by a monotonous pattern are decidable properties on the class of backward-effective wqo-SDN with monotonous patterns and queries.*

*Proof.* The termination of a case associated with an identifier $id$ is equivalent to the coverability of the marking with one token $(D_\perp, id)$ in place $p_{out}$ (and all other places empty) by the marking resulting from the initialization of this case (using transition $t_{in}$) where $D_\perp$ is the least document (reduced to an untagged root).

Decidability of soundness also stems from decidability of coverability. An SDN is sound if it terminates and whenever place $p_{out}$ contains a token, one cannot find another place containing a token with the same identifier, i.e. for each place $p \in T \setminus \{t_{out}\}$ the marking $M_p$ with token $(D_\perp, id)$ in both place $p_{out}$ and $p$ and with no other tokens is coverable from the initial marking.

Coverability, termination and soundness have solutions for a single given initial marking, i.e. for a particular chosen case. We would like to consider whether transactions terminate or cover a given marking $M$ for all or for some possible inputs to the system. We suppose that the set of results output by query $\langle t_{in}, p_{in}\rangle$ is the **symbolic set of documents** from $\mathrm{Doc}_{\tau,\leq n}$ that satisfy a particular monotonous pattern $P$. Then, one can compute the set $BSat(P)$ of documents obtained by replacing ancestor edges of $P$ by sequences of edges with untagged nodes in such a way that the depth of the obtained document remains smaller than $n$, and replacing each constraint $\gamma$ on values attached to a node of $P$ by a value selected from a basis for the upward closed set of values satisfying $\gamma$. This basis exists as $P$ is monotonous, and the domains are wqos. This set $BSat(P)$ forms a basis for all documents embedding $P$. Noticing that $\mathcal{R}(M) \leq \mathcal{R}(M')$ when $M \leq M'$ for wqo and backward effective SDNs with monotonous queries and patterns, coverability and termination can be verified for all cases initiated by $\langle t_{in}, p_{in}\rangle$ if it can be proved for all elements in $BSat(P)$. Note that it is sufficient to compute the fixpoint returned by the backward coverability algorithm and then compare this set with all minimal elements in $BSat(P)$. □

The above decidability results do not extend to reachability:

**Theorem 3 (Undecidability of reachability).** *Reachability is undecidable, even for backward effective wqo-SDN with monotonous patterns and queries.*

*Proof.* An SDN can easily simulate reset Petri nets for which reachability is undecidable [12]. We need only to deal with place $p_{in}$ (in order to conform with Definition 6 we can assume a transition $t$ from $p_{in}$ to $p_{out}$ such that pattern $\langle p_{in}, t\rangle$ is never satisfied). The content of place $p_{in}$ encodes a particular marking of the reset net: A document $D$ with a root node and a child labeled $p$ with $n$ children indicates that place $p$ of the reset net contains $n$ tokens. This

set of tokens can be manipulated as a whole, incremented or decremented by monotonous queries. Enabledness of a transition can be encoded by a pattern that tests the existence of a token in some place $p$, i.e. they are trees with a root, a child node tagged $p$ and one children. Monotonous queries can be used to increment or decrement the number of children of a particular node tagged by $p$, encoding consumption or creation of tokens. Last, a query can remove all children of a document, simulating a reset arc. Such queries are monotonous, and transitions using this kind of queries are also backward effective. It is also obvious that one can design a transition that will fire only once, produce a set of documents encoding the initial marking of a reset net, and will then ignore all transactions produced by $\langle t_{in}, p_{in} \rangle$. Undecidability of the reachability problem for reset nets [12] concludes the proof. □

This negative result should not be seen as a severe limitation: reachability is usually undecidable outside the class of Petri nets, and when considering transactional systems, properties of interest are usually not expressed in terms of global states.

**Proposition 4.** *Well quasi orderedness of an SDN is undecidable. Coverability, reachability and termination problems are undecidable for wqo SDNs.*

*Proof.* We design a wqo SDN that encodes a two counters machine. A two counters machine is given as a pair of counters $C_1, C_2$ holding non-negative integers and a finite list of instructions $l_1, \dots l_n$ each of which, except the last one, is of one of the following forms: *i)* $l_i : inc(C_\ell)$ meaning that we increment counter $C_\ell$ and then go to the following instruction, *ii)* $l_j$ : **if** $(C_\ell = 0)$, $l_k$ **else** $dec(C_j)$, $l_{k'}$ indicating that if counter $C_\ell$ is null we must proceed to instruction $l_k$ otherwise we decrement this counter and go to instruction $l_{k'}$. The machine halts when it reaches the last instruction $l_n : Halt$. A configuration of a counter machine is given by the value of its counters, and the current instruction line. The machine usually starts at instruction 0, with counters set to 0. It is well-known that one can not decide if a counter machine halts. For any counter machine, we can define an SDN (represented in Figure 6) that encodes the moves of this machine.

First, we can encode a counter machine configuration as a document with three nodes: a root, and its left and right children. The root is tagged by an instruction number from $l_1, \dots, l_n$, the left and right children are tagged by $c_1$ and $c_2$ respectively with values given by non-negative integers. The corresponding documents are of bounded depth with values from wqo domains. For each instruction of the form $l_i : inc(C_\ell)$, we design a transition $t_i$ with ${}^\bullet t_i = t_i^\bullet = p_{in}$ such that $P_i = \langle p, t_i \rangle$ is the pattern reduced to a root whose tag has value $l_i$ and $Q_i = \langle t_i, p \rangle$ is the query that transforms a document into a document with root $l_{i+1}$, and such that the value attached to the node with tag $c_\ell$ is incremented by one, and the other one is left unmodified. For each instruction of the form $l_j$ : **if** $(C_\ell = 0)$, $l_k$ **else** $dec(C_j)$, $l_{k'}$ we design two transitions $t_{j,Z}$ and $t_{j,NZ}$ such that $P_{j,Z} = \langle p_{in}, t_{j,Z} \rangle$ is a pattern testing if the root of a document is labeled by $l_j$, and the value of node with tag $c_\ell$ is zero, $Q_{j,Z} = \langle t_{j,Z}, p_{in} \rangle$ is the query that transforms a document into a document with root $l_k$, and such that the values

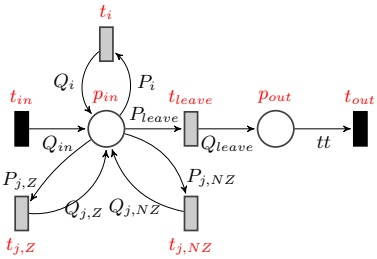



**Fig. 6.** Encoding a counter Machine with wqo Structured Data Nets

attached to child nodes remain unchanged, $P_{j,NZ} = \langle p_{in}, t_{j,NZ} \rangle$ is the pattern testing if the root of a document is labeled by $l_j$, and the value of node with tag $c_\ell$ is greater than zero, $Q_{j,NZ} = \langle t_{j,NZ}, p_{in} \rangle$ is the query that transforms a document into a document with root $l_{k'}$, and such that the value attached to node with tag $c_\ell$ is decremented by one and the value attached to the other child node remains unchanged. The initial configuration of the counter machine is created by query $Q_{in} = \langle t_{in}, p_{in} \rangle$ that produces a document with root labeled $l_0$ and two children nodes tagged respectively by $c_1, c_2$ with values 0. We set $M_0$ as an initial marking in which all places are empty. Transition $t_{leave}$ moves the token from place $p$ to $p_{out}$ if the root tag has value $l_n$, i.e. the machine halted. Clearly, the counter machine terminates iff one can reach a configuration in which $p_out$ is not empty. Thus one cannot decide termination, and similarly the reachability or coverability (of the marking with just one token in $p_{out}$).

Let us now prove that one can not decide whether a net is wqo. One can add a transition $t_{nobnd}$ to the above net such that ${}^\bullet t_{nobnd} = t_{nobnd}^\bullet = p_{out}$, $\langle p_{out}, t_{nobnd} \rangle = tt$, and $\langle t_{nobnd}, p_{out} \rangle$ is a query that increases the depth of a document by 1, by inserting a children with some tag $a$ between the root and its first child (hence creating successive incomparable documents). Then the counter machine terminates iff the corresponding SDN is not wqo. □

Even though well-quasi-orderedness of a net is undecidable, one still can rely on restriction on queries to ensure this property. In many systems, queries are used to extract data from a dataset (a list of records). The result is also a list of records that can be again assembled as a bounded depth document. Other queries compute new values from datasets (sums, means, etc. ) and insert the results is a new document (a "form") of bounded depth and size. So, one can restrict to queries that produce only documents of bounded depth, which values domains are finite sets or wqo sets such as integers without harming too much the expressiveness of the model. Form filling queries that manipulate integers, rationals or strings are also very often backward effective, provided the mechanisms used to select the nodes carrying the values of interest to fill a form are monotonous.

## 6 Conclusion

This paper has addressed an extension of Petri Nets which transitions manipulate structured data via patterns and queries. Without limitations, this model is

Turing Powerful. However, under some restrictions on the nature of queries and on the shape of documents, interesting properties such as coverability are decidable. We believe that limiting data to semi-structured documents of bounded depth with wqo labels is a sensible approach: many information systems use strings, booleans, etc, but do not need real values with arbitrary precision. The major limitation w.r.t documents is that data structures such as files and queues can not be encoded with our model.

Several improvements might be investigated. First, the coverability proof relies on backward effectiveness of transitions to guarantee effectiveness of the WSTS associated to a wqo SDN with monotonous queries and patterns. This does not identify a particular class of queries. To be practical, we would like to identify classes of non-trivial monotonous queries that ensure effectiveness. Decidability results for positive active XML [3] use another form of monotonicity: they assume that a document can only grow, which can be an adequate assumption in case management systems. Considering positive SDN could be a way to ensure effectiveness. Another improvement lies in pattern expressiveness: currently, only individual constraints on data values are attached to nodes. One could, however, consider patterns with constraints of the form $v.\sigma \leq v'.\sigma'$, involving values of several nodes, sets of patterns requiring matching on several documents from a place, boolean combinations of patterns,... and see how these extensions affect positive results. Another line of research concerns symbolic manipulation of upward closed sets of documents. So far, we have considered coverability for symbolic set of initial cases, but we can imagine to define symbolic sets of initial markings, database contents, or target markings to cover. Last, we could consider extensions of the model with some essential features for web services and transactional systems, for example allowing for transaction cancellation. Such feature is currently not handled by our model: one can even remark that an SDN might not be sound, even when it is wqo and backward effective. Actually, such an extension would mean adding features from reset Petri nets to our model. Since coverability is decidable for reset Petri nets, this extension could preserve decidability results obtained in this paper.

## References


1. P.A. Abdulla, K. Cerans, B. Jonsson, and Y-K Tsay. General decidability theorems for infinite-state systems. In *Proc. of LICS'96*, pages 313–321. IEEE, 1996.
2. S. Abiteboul, O. Benjelloun, I. Manolescu, T. Milo, and R. Weber. Active XML: A Data-Centric Perspective on Web Services. In *BDA02*, 2002.
3. S. Abiteboul, O. Benjelloun, and T. Milo. Positive active XML. In *Proc. of PODS'04*, pages 35–45. ACM, 2004.
4. L. Acciai and M. Boreale. Deciding safety properties in infinite-state pi-calculus via behavioural types. In *ICALP (2)*, volume 5556 of *LNCS*, pages 31–42, 2009.
5. S. Akshay, L. Hélouet, and M. Mukund. Sessions with an unbounded number of agents. In *ACSD'14*, volume 4281, pages 166–175. IEEE, 2014.
6. T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. Business process execution language for Web services (BPEL4WS). version 1.1, 2003.

7. E. Badouel, L. Hélouët, G.-E. Kouamou, and C. Morvan. A grammatical approach to data-centric case management in a distributed collaborative environment. In *SAC'15*, Salamanca, Spain, April 2015. ACM.
8. M. Boreale, R. Bruni, R. De Nicola, and M. Loreti. Sessions and pipelines for structured service programming. In *FMOODS*, volume 5051 of *LNCS*, pages 19–38, 2008.
9. R. Bruni, I. Lanese, H.C. Melgratti, and E. Tuosto. Multiparty sessions in SOC. In *COORDINATION*, volume 5052 of *LNCS*, pages 67–82, 2008.
10. E. Damaggio, A. Deutsch, and V. Vianu. Artifact systems with data dependencies and arithmetic. *ACM Trans. Database Syst.*, 37(3):22, 2012.
11. G. Ding. Subgraphs and well-quasi-ordering. In *Journal of Graph Theory*, volume 16(5), pages 489 – 502, 1992.
12. C. Dufourd, A. Finkel, and Ph. Schnoebelen. Reset nets between decidability and undecidability. In *Proc. of ICALP'98*, volume 1443 of *LNCS*, pages 103–115. Springer, 1998.
13. A. Finkel and Ph. Schnoebelen. Well-structured transition systems everywhere! *Theor. Comput. Sci.*, 256(1-2):63–92, 2001.
14. B. Genest, A. Muscholl, and Z Wu. Verifying recursive active documents with positive data tree rewriting. In *Proc. of FSTTCS 2010*, volume 8 of *LIPIcs*, pages 469–480. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2010.
15. G. Higman. Ordering by divisibility in abstract algebras. *Proc. London Math. Soc. (3)*, 2:326–336, 1952.
16. K. Honda, N. Yoshida, and M. Carbone. Multiparty asynchronous session types. In *POPL*, pages 273–284, 2008.
17. R. Hull, E. Damaggio, F. Fournier, M. Gupta, F.T. Heath, S. Hobson, M.H. Linehan, S. Maradugu, A. Nigam, P. Sukaviriya, and R. Vaculín. Introducing the guard-stage-milestone approach for specifying business entity lifecycles. In *Proc. of WS-FM 2010*, volume 6551 of *LNCS*, pages 1–24. Springer, 2011.
18. K. Jensen. *Coloured Petri Nets - Basic Concepts, Analysis Methods and Practical Use - Volume 1, Second Edition.* Monographs in Theoretical Computer Science. An EATCS Series. Springer, 1996.
19. R. Lazic, Tom Newcomb, J. Ouaknine, A.W. Roscoe, and J. Worrell. Nets with tokens which carry data. *Fundam. Inform.*, 88(3):251–274, 2008.
20. J. Misra and W. Cook. Computation orchestration. *Software and Systems Modeling*, 6(1):83–110, 2007.
21. I. Mlynkova, K. Toman, and J. Pokorný. Statistical analysis of real XML data collections. In *Proc. of International Conference on Management of Data'06*, pages 15–26. Tata McGraw-Hill, 2006.
22. A. Nigam and N. S. Caswell. Business artifacts: An approach to operational specification. *IBM Syst. J.*, 42:428–445, July 2003.
23. OASIS. Web Services Business Process Execution Language. Technical report, OASIS, 2007. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf.
24. R. Pugliese and F. Tiezzi. A calculus for orchestration of Web services. *J. Applied Logic*, 10(1):2–31, 2012.
25. T. Wies, T. Zufferey, and T.A. Henzinger. Forward analysis of depth-bounded processes. In *FOSSACS*, volume 6014 of *LNCS*, pages 94–108, 2010.
26. World Wide Web Consortium. XML path language (xpath). Technical report, W3C, 1999. W3C Recommendation,http://www.w3.org/TR/xpath.
27. World Wide Web Consortium. XQuery 1.0: An XML Query Language. Technical report, W3C, 1999. W3C Recommendation,http://www.w3.org/TR/xquery.